package com.twitter.frigate.pushservice.adaptor

import com.twitter.finagle.stats.Stat

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.\_

import com.twitter.frigate.common.candidate.\_

import com.twitter.frigate.common.predicate.CommonOutNetworkTweetCandidatesSourcePredicates.filterOutReplyTweet

import com.twitter.frigate.pushservice.model.PushTypes.RawCandidate

import com.twitter.frigate.pushservice.model.PushTypes.Target

import com.twitter.frigate.pushservice.params.PushFeatureSwitchParams

import com.twitter.frigate.pushservice.params.PushParams

import com.twitter.frigate.pushservice.util.PushDeviceUtil

import com.twitter.hermit.store.tweetypie.UserTweet

import com.twitter.recos.recos\_common.thriftscala.SocialProofType

import com.twitter.search.common.features.thriftscala.ThriftSearchResultFeatures

import com.twitter.stitch.tweetypie.TweetyPie.TweetyPieResult

import com.twitter.storehaus.ReadableStore

import com.twitter.timelines.configapi.Param

import com.twitter.util.Future

import com.twitter.util.Time

import scala.collection.Map

case class EarlyBirdFirstDegreeCandidateAdaptor(

earlyBirdFirstDegreeCandidates: CandidateSource[

EarlybirdCandidateSource.Query,

EarlybirdCandidate

],

tweetyPieStore: ReadableStore[Long, TweetyPieResult],

tweetyPieStoreNoVF: ReadableStore[Long, TweetyPieResult],

userTweetTweetyPieStore: ReadableStore[UserTweet, TweetyPieResult],

maxResultsParam: Param[Int],

globalStats: StatsReceiver)

extends CandidateSource[Target, RawCandidate]

with CandidateSourceEligible[Target, RawCandidate] {

type EBCandidate = EarlybirdCandidate with TweetDetails

private val stats = globalStats.scope("EarlyBirdFirstDegreeAdaptor")

private val earlyBirdCandsStat: Stat = stats.stat("early\_bird\_cands\_dist")

private val emptyEarlyBirdCands = stats.counter("empty\_early\_bird\_candidates")

private val seedSetEmpty = stats.counter("empty\_seedset")

private val seenTweetsStat = stats.stat("filtered\_by\_seen\_tweets")

private val emptyTweetyPieResult = stats.stat("empty\_tweetypie\_result")

private val nonReplyTweetsCounter = stats.counter("non\_reply\_tweets")

private val enableRetweets = stats.counter("enable\_retweets")

private val f1withoutSocialContexts = stats.counter("f1\_without\_social\_context")

private val userTweetTweetyPieStoreCounter = stats.counter("user\_tweet\_tweetypie\_store")

override val name: String = earlyBirdFirstDegreeCandidates.name

private def getAllSocialContextActions(

socialProofTypes: Seq[(SocialProofType, Seq[Long])]

): Seq[SocialContextAction] = {

socialProofTypes.flatMap {

case (SocialProofType.Favorite, scIds) =>

scIds.map { scId =>

SocialContextAction(

scId,

Time.now.inMilliseconds,

socialContextActionType = Some(SocialContextActionType.Favorite)

)

}

case (SocialProofType.Retweet, scIds) =>

scIds.map { scId =>

SocialContextAction(

scId,

Time.now.inMilliseconds,

socialContextActionType = Some(SocialContextActionType.Retweet)

)

}

case (SocialProofType.Reply, scIds) =>

scIds.map { scId =>

SocialContextAction(

scId,

Time.now.inMilliseconds,

socialContextActionType = Some(SocialContextActionType.Reply)

)

}

case (SocialProofType.Tweet, scIds) =>

scIds.map { scId =>

SocialContextAction(

scId,

Time.now.inMilliseconds,

socialContextActionType = Some(SocialContextActionType.Tweet)

)

}

case \_ => Nil

}

}

private def generateRetweetCandidate(

inputTarget: Target,

candidate: EBCandidate,

scIds: Seq[Long],

socialProofTypes: Seq[(SocialProofType, Seq[Long])]

): RawCandidate = {

val scActions = scIds.map { scId => SocialContextAction(scId, Time.now.inMilliseconds) }

new RawCandidate with TweetRetweetCandidate with EarlybirdTweetFeatures {

override val socialContextActions = scActions

override val socialContextAllTypeActions = getAllSocialContextActions(socialProofTypes)

override val tweetId = candidate.tweetId

override val target = inputTarget

override val tweetyPieResult = candidate.tweetyPieResult

override val features = candidate.features

}

}

private def generateF1CandidateWithoutSocialContext(

inputTarget: Target,

candidate: EBCandidate

): RawCandidate = {

f1withoutSocialContexts.incr()

new RawCandidate with F1FirstDegree with EarlybirdTweetFeatures {

override val tweetId = candidate.tweetId

override val target = inputTarget

override val tweetyPieResult = candidate.tweetyPieResult

override val features = candidate.features

}

}

private def generateEarlyBirdCandidate(

id: Long,

result: Option[TweetyPieResult],

ebFeatures: Option[ThriftSearchResultFeatures]

): EBCandidate = {

new EarlybirdCandidate with TweetDetails {

override val tweetyPieResult: Option[TweetyPieResult] = result

override val tweetId: Long = id

override val features: Option[ThriftSearchResultFeatures] = ebFeatures

}

}

private def filterOutSeenTweets(seenTweetIds: Seq[Long], inputTweetIds: Seq[Long]): Seq[Long] = {

inputTweetIds.filterNot(seenTweetIds.contains)

}

private def filterInvalidTweets(

tweetIds: Seq[Long],

target: Target

): Future[Seq[(Long, TweetyPieResult)]] = {

val resMap = {

if (target.params(PushFeatureSwitchParams.EnableF1FromProtectedTweetAuthors)) {

userTweetTweetyPieStoreCounter.incr()

val keys = tweetIds.map { tweetId =>

UserTweet(tweetId, Some(target.targetId))

}

userTweetTweetyPieStore

.multiGet(keys.toSet).map {

case (userTweet, resultFut) =>

userTweet.tweetId -> resultFut

}.toMap

} else {

(target.params(PushFeatureSwitchParams.EnableVFInTweetypie) match {

case true => tweetyPieStore

case false => tweetyPieStoreNoVF

}).multiGet(tweetIds.toSet)

}

}

Future.collect(resMap).map { tweetyPieResultMap =>

val cands = filterOutReplyTweet(tweetyPieResultMap, nonReplyTweetsCounter).collect {

case (id: Long, Some(result)) =>

id -> result

}

emptyTweetyPieResult.add(tweetyPieResultMap.size - cands.size)

cands.toSeq

}

}

private def getEBRetweetCandidates(

inputTarget: Target,

retweets: Seq[(Long, TweetyPieResult)]

): Seq[RawCandidate] = {

retweets.flatMap {

case (\_, tweetypieResult) =>

tweetypieResult.tweet.coreData.flatMap { coreData =>

tweetypieResult.sourceTweet.map { sourceTweet =>

val tweetId = sourceTweet.id

val scId = coreData.userId

val socialProofTypes = Seq((SocialProofType.Retweet, Seq(scId)))

val candidate = generateEarlyBirdCandidate(

tweetId,

Some(TweetyPieResult(sourceTweet, None, None)),

None

)

generateRetweetCandidate(

inputTarget,

candidate,

Seq(scId),

socialProofTypes

)

}

}

}

}

private def getEBFirstDegreeCands(

tweets: Seq[(Long, TweetyPieResult)],

ebTweetIdMap: Map[Long, Option[ThriftSearchResultFeatures]]

): Seq[EBCandidate] = {

tweets.map {

case (id, tweetypieResult) =>

val features = ebTweetIdMap.getOrElse(id, None)

generateEarlyBirdCandidate(id, Some(tweetypieResult), features)

}

}

/\*\*

\* Returns a combination of raw candidates made of: f1 recs, topic social proof recs, sc recs and retweet candidates

\*/

def buildRawCandidates(

inputTarget: Target,

firstDegreeCandidates: Seq[EBCandidate],

retweetCandidates: Seq[RawCandidate]

): Seq[RawCandidate] = {

val hydratedF1Recs =

firstDegreeCandidates.map(generateF1CandidateWithoutSocialContext(inputTarget, \_))

hydratedF1Recs ++ retweetCandidates

}

override def get(inputTarget: Target): Future[Option[Seq[RawCandidate]]] = {

inputTarget.seedsWithWeight.flatMap { seedsetOpt =>

val seedsetMap = seedsetOpt.getOrElse(Map.empty)

if (seedsetMap.isEmpty) {

seedSetEmpty.incr()

Future.None

} else {

val maxResultsToReturn = inputTarget.params(maxResultsParam)

val maxTweetAge = inputTarget.params(PushFeatureSwitchParams.F1CandidateMaxTweetAgeParam)

val earlybirdQuery = EarlybirdCandidateSource.Query(

maxNumResultsToReturn = maxResultsToReturn,

seedset = seedsetMap,

maxConsecutiveResultsByTheSameUser = Some(1),

maxTweetAge = maxTweetAge,

disableTimelinesMLModel = false,

searcherId = Some(inputTarget.targetId),

isProtectTweetsEnabled =

inputTarget.params(PushFeatureSwitchParams.EnableF1FromProtectedTweetAuthors),

followedUserIds = Some(seedsetMap.keySet.toSeq)

)

Future

.join(inputTarget.seenTweetIds, earlyBirdFirstDegreeCandidates.get(earlybirdQuery))

.flatMap {

case (seenTweetIds, Some(candidates)) =>

earlyBirdCandsStat.add(candidates.size)

val ebTweetIdMap = candidates.map { cand => cand.tweetId -> cand.features }.toMap

val ebTweetIds = ebTweetIdMap.keys.toSeq

val tweetIds = filterOutSeenTweets(seenTweetIds, ebTweetIds)

seenTweetsStat.add(ebTweetIds.size - tweetIds.size)

filterInvalidTweets(tweetIds, inputTarget)

.map { validTweets =>

val (retweets, tweets) = validTweets.partition {

case (\_, tweetypieResult) =>

tweetypieResult.sourceTweet.isDefined

}

val firstDegreeCandidates = getEBFirstDegreeCands(tweets, ebTweetIdMap)

val retweetCandidates = {

if (inputTarget.params(PushParams.EarlyBirdSCBasedCandidatesParam) &&

inputTarget.params(PushParams.MRTweetRetweetRecsParam)) {

enableRetweets.incr()

getEBRetweetCandidates(inputTarget, retweets)

} else Nil

}

Some(

buildRawCandidates(

inputTarget,

firstDegreeCandidates,

retweetCandidates

))

}

case \_ =>

emptyEarlyBirdCands.incr()

Future.None

}

}

}

}

override def isCandidateSourceAvailable(target: Target): Future[Boolean] = {

PushDeviceUtil.isRecommendationsEligible(target)

}

}